Self-adaptive mutation in on-line, on-board evolutionary robotics

A.E. Eiben, Giorgos Karafotias, Evert Haasdijk
Dept. of Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
Email: {gusz,gks290,e.haasdijk}@few.vu.nl

***Abstract*—On-line, on-board evolution of robot controllers implies an inherent need for adjusting the parameters of the evolutionary algorithm on-the-fly. In this paper we argue that the most influential factor to govern evolution in our application is the mutation operator. To address the problem of adjusting its parameter(s) we identify different on-line parameter control mechanisms and perform an experimental comparison among them. The experiments are carried out in a high quality simulator, Webots, for three different tasks for the robots. The results are not fully consistent over the tasks considered, yet they support a preference for the de-randomised self-adaptive mutation step size control.**

I. Background and objectives

The work presented in this paper is grounded in a European research project, called SYMBRION: Symbiotic Evolutionary Robot Organisms[1]. SYMBRION embraces pervasive adaptation through evolution: individual robots are fully autonomous and viable, evolving their controllers on-the-fly to adapt autonomously to their environments and tasks. To particularise the type of evolution we need here, we use the terms on-line and on-board, where "on-line" indicates that robots' controllers are evolving during (and not before) their operational period, while "on-board" shows that the computational processes behind evolution all take place inside (and not outside) the robots.

In our previous work [3] we have discussed on-line, on-board evolution in detail, positioning it with respect to mainstream evolutionary robotics [14]. To this end, we set up a classification scheme based on a set of three features concerning the evolution of controllers from temporal, spatial, and procedural perspective. That is, we distinguish types of evolution considering when it happens, where it happens, and how it happens, cf. [3]:

1) off-line or design time vs. on-line or run time (when),
2) on-board or intrinsic vs. off-board or extrinsic (where),
3) in an encapsulated or centralised vs. distributed manner (how).

Note, that we do not include embodiedness (of fitness evaluations) in this classification scheme. The reason is that the system we have in mind is one with real robots, where fitness evaluation always happens in reality. In other words, our whole scheme falls in the category of embodied evolution in terms of the classification scheme offered by Watson *et al.* in [21].

The traditional approach to evolutionary robotics (ER) is to use a conventional evolutionary algorithm (EA) for finding good controllers in a fashion that can be identified as off-line, off-board (extrinsic), and centralised (encapsulated in an external computer). The population of genotypes that encode for robot controllers is evolved on a computer that executes the evolutionary operators (variation and selection), managed in a centralised fashion. As for fitness evaluation, the computer can invoke a simulator or send the genotype to be evaluated to a robot to test it (embodied evaluation). Evaluation of genotypes can happen in parallel or one by one. At the end of the evolutionary process the controller encoded by the best genotype is deployed in the robot(s).

The encapsulated evolution approach is on-line, on-board (intrinsic), and centralised. Each robot has an evolutionary algorithm implemented on-board, maintaining a population of genotypes inside itself. The robots run these (possibly different) evolutionary algorithms locally and perform the fitness evaluations autonomously. This is typically done in a time-sharing system, where one member of the inner population is activated (i.e., decoded into a controller) at a time and is used for a while to gather feedback on its quality. Here, the iterative improvement (optimisation) of controllers is the result of the evolutionary algorithms running in parallel on the individual robots [20].

The distributed evolution approach can be described as on-line, on-board (intrinsic), and distributed. Each robot has a single genotype and is controlled by the corresponding phenotype. Robots can reproduce autonomously and asynchronously and create offspring controllers by recombining and/or mutating their genotypes. Here, the iterative improvement (optimisation) of controllers is the result of the evolutionary process that emerges from the exchange of genetic information among the robots [21].

The latter two approaches can be combined, resulting in a set-up akin to an island model as used in parallel genetic algorithms. In such a combined system, each robot is an island and genetic information is exchanged through intra-island variation (i.e., within the population of the

[1]EU grant agreement 216342, www.symbrion.eu

encapsulated evolutionary algorithm in one robot) and inter-island migration (between two or more robots) [13], [19], [22], [16], [6], [8].

The present paper is concerned with the encapsulated evolution approach, where each robot autonomously runs its own evolutionary algorithm inside itself. In principle, this EA can be of any type, as long as it matches the representation of the robot controllers. To clarify this, consider the difference between a robot controller and the evolvable code that represents that controller. In general, a robot controller is a structurally and procedurally complex entity that directly determines the robots behaviour. When using evolutionary methods for controller design, controllers are perceived as phenotypes that are represented by relatively simple pieces of code, called genotypes. A formal definition of the representation requires a possibly complex mapping from genotype to phenotype. The fitness of a robot controller (typically: task performance) is then determined by the phenotype. Meanwhile, –conforming to the principles of biological evolution– it is only the genotypes that undergo evolutionary operators, mutation and/or crossover, not the phenotypes. In general, a robot can contain more than one genotype, but at all times only *one* of these can be active, i.e., decoded into a working phenotype/controller. Figure 1 illustrates this matter.

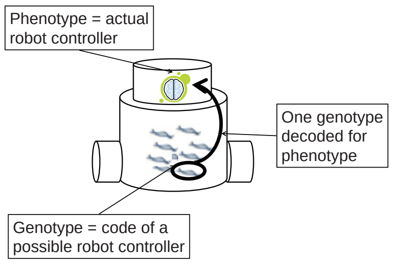

Phenotype = actual robot controller
One genotype decoded for phenotype
Genotype = code of a possible robot controller

Figure 1. Encapsulated evolution, where each robot is running its own (centralised) evolutionary algorithm on-board.

It is obvious that using encapsulated evolution in an autonomous fashion, where no human intervention is possible, the evolutionary algorithm in the robots must be able to optimise under unforeseen and possibly very different conditions. Unfortunately, the performance of evolutionary algorithms is, in general, quite dependent on their settings. Hence, such a system requires an evolutionary algorithm that is capable of calibrating itself on-the-fly. In usual evolutionary computing terms this means that we need evolutionary algorithms with good parameter control mechanisms [4].

Let us note that changing the value of some parameter X in an EA, e.g., population size or mutation rate, can have a much bigger impact on EA behaviour than changing the value of another parameter Y, making parameter X more relevant for the EA behaviour than parameter Y. Thus, in general, the relevance of the parameters of an EA can vary considerably [12], [18]. Moderately relevant parameters need not be calibrated very carefully: an educated guess for a good value is likely to work under many circumstances. For our on-line, on-board evolutionary robotics application this means that moderately relevant parameters can be used with fixed settings, but we need good parameter control mechanisms for the most relevant parameters. Of course, the relevance of a parameter depends on the given EA and the problem to be solved. However, over the history of the evolutionary computing field there are numerous indications that parameters related to the mutation operator are the most relevant, which led to powerful mutation control mechanisms within evolution strategies [1], [11], [17].

The main objective of this paper is to investigate two mechanisms to control mutation in an on-line, on-board evolutionary application, based on the encapsulated approach. One of them is the standard self-adaptation mechanism within evolution strategies that has known problems with very small populations that we have here [10]. The other one is the de-randomised self-adaptation mechanism that has been offered to circumvent these problems [15].

II. The $(\mu + 1)$ on-line Evolutionary Algorithm and its parameters

In previous publications we have introduced an encapsulated EA to be used for on-line, on-board evolution of robot controllers, where a population of μ individuals is maintained within each robot [2]. This algorithm is motivated by a number of considerations regarding the type of application we face here. First, the very noisy character of fitness calculations, caused by the fact that "newborn" controllers start at different positions – in fact, where the previous controller left the robot. Second, the very low computational budget in terms of search steps, where a search step is the creation and evaluation of a new controller. This budget is low, because a robot cannot permit wasting time on bad controllers and has to achieve a satisfactory level of performance in real-time. To cope with noisy fitness we use re-evaluations. To prevent wasting too much time on poor candidate solutions we limit the number of child controllers to one in each cycle and use "pushy" evolutionary operators. The main outlines of the resulting algorithm can be listed as follows:

- In every evolutionary cycle two things can happen: either an existing individual is re-evaluated by re-activating it for τ time steps (with probability ρ), or a new individual is generated and evaluated (with probability $1 - \rho$).
- When a new child controller needs to be created, two parents are chosen based on their fitness in a nondeterministic way.
- A new child controller is created by recombining the two parents and mutating the result.
- A new controller is immediately activated and evaluated as follows:

  - Before the real evaluation period starts, the robot can spend some time (the recovery period) without its fitness being measured.
  - During the evaluation period (τ time steps) the robot is active and its fitness is being measured.
- After the new controller is evaluated it has to compete with the existing ones for a place in the population, based on its fitness.

This algorithmic core has been adjusted for a case study where controllers were represented by real-valued vectors [2]. With this addition we can specify crossover and mutation operators (that are always representation-dependent). We have chosen to use arithmetic averaging recombination and mutation by adding noise from a Gaussian distribution with zero mean. Furthermore, we use binary tournaments for selecting good parents. For a full description of the resulting algorithm, we refer to [7].

There are four main parameters of this algorithm:

ρ	The re-evaluation rate;
τ	The duration of controller evaluation;
μ	The population size;
σ	The mutation step-size.

In a previous study we have investigated these parameters, in particular, we looked into their relevance, i.e., their impact on the performance of the EA [7]. There we concluded that the mutation step-size σ is by far the most influential one. The values of the other parameters did not seem to matter that much (on the problem considered there). It should be noted that the $(\mu + 1)$ ON-LINE algorithm is new and so is the on-line, on-board evolutionary robotics system it is implemented in. Therefore, there are not many (empirical) results showing its behavior. Hence, it is too early to formulate generic statements about it, but we are confident that the main finding in [7] regarding the mutation step-size is generally valid.

III. Tasks and robot description

In this section we describe the robot setup and the test problems we used in our experiments. The tasks used in the experiments are fast forward, phototaxis, and resource gathering. The details are given below.

Fast forward: The first task is frequently used in evolutionary robotics and requires the robot to move fast and straight-ahead but also avoid obstacles, implying a trade-off between forward movement and obstacle avoidance [14]. For this task the robot uses 8 proximity sensors. Task performance, the fitness measure used in the evolutionary algorithm, is given by Eq. 1. The arena in which the robot moves is shown in Figure 2(a).

$$f = \sum_{t=0}^{\tau} (v_t \cdot (1 - v_r) \cdot (1 - d)) \tag{1}$$

where v_t and v_r are the translational and the rotational speed, respectively. v_t is normalised between -1 (full speed reverse) and 1 (full speed forward), v_r between 0 (movement in a straight line) and 1 (maximum rotation); d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle).

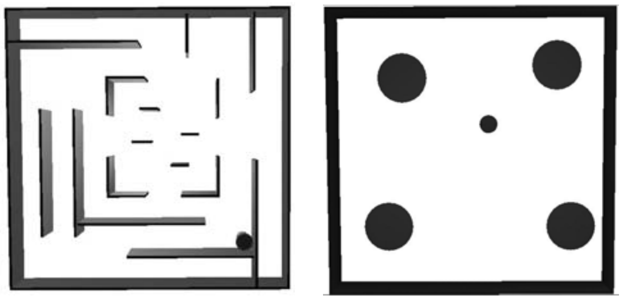

(a) The arena for the fast forward task. The circle represents the e-puck robot to scale.
(b) The arena for the resource gathering task. The small circle represents the e-puck robot to scale. The larger circles represent the chargers' active areas to scale: once the robot enters the area of a charger (that is currently full) it instantly collects the energy.

Figure 2. The tasks.

Phototaxis: For this task the robot has to move towards a stationary light source and remain close to it. The input to the neural net is the set of 8 light sensors of the e-puck. The fitness function is given by Eq. 2. The arena is simply an empty square with a light source in the centre.

$$f = \sum_{t=0}^{\tau} maxSensorValue; \tag{2}$$

Resource gathering: In this task the robot has to move around in an empty square arena and collect "resources", i.e. energy from chargers. Each charger is indicated with a light above it. When the robot moves to the charger it collects the energy and the light is turned off. The charger slowly replenishes and, when full again, its light is turned on. For this task, the robot's eight light sensors are used. The fitness function is given in Eq. 3. The arena with the chargers' positions is shown in Figure 2(b).

$$f = \sum_{t=0}^{\tau} collectedEnergy; \tag{3}$$

The experiments were performed with a simulated e-puck[2] robot using the Webots[3] environment. As for the robot controller, we use a straightforward perceptron neural net with hyperbolic tangent activation function. The neural net has 9 input nodes (8 sensor inputs and a bias node), no hidden nodes and 2 output nodes (the left and right motor values), resulting in a total of 18 weights.

[2]http://www.e-puck.org/
[3]http://www.cyberbotics.com/

IV. Algorithm setup

Here we describe the setup of the $(\mu + 1)$ ON-LINE algorithm for our experiments. To represent the robot controllers (phenotypes) we have chosen real-valued vectors with 18 variables with the obvious mapping: each variable corresponds to one of the 18 weights of the neural net. This implies that the $(\mu + 1)$ ON-LINE evolutionary algorithm needs to optimise real-valued vectors of length 18. To this end we use the variation operators arithmetic averaging recombination and mutation by adding a random number drawn from a Gaussian distribution with zero mean and standard deviation σ, where σ is called the mutation step-size.

The main objective of this paper is to compare different control strategies for regulating mutation on-the-fly. In particular, we use self-adaptation of mutation step-sizes, as usual in evolution strategies [1], [10], [17]. The essence of this method is to encode mutation step-sizes into the genotypes and co-evolve their values together with the ones encoding for the 'real' solution – here: the weights of the neural nets. We have chosen to use an individual mutation step-size σ_i for each variable x_i and have extended the representation accordingly. This means that the evolutionary algorithm works on vectors of 36 variables $\langle x_1, \dots, x_{18}, \sigma_1, \dots, \sigma_{18}\rangle$, instead of vectors of the form $\langle x_1, \dots, x_{18}\rangle$. The recombination operator works the same on both the $\bar{x}$-part and the $\bar{\sigma}$-part. However, mutating the $\bar{x}$-part and the $\bar{\sigma}$-part is different. The $\bar{x}$-part is mutated by the usual $x'_i = x_i + N(0, \sigma)$, where $N(0, \sigma)$ stands for a random number drawn from the normal distribution.

As for mutating the mutation step-sizes we use the following two methods.

1) *Multi-step Self-Adaptation:* This is pretty much the standard in evolution strategies, where the individual σs are updated as follows:

$$\sigma'_i = \sigma_i \cdot e^{N(0,\tau') + N_i(0,\tau)}$$

where $\tau' = 1/\sqrt{2n}$, and $\tau = 1/\sqrt{2\sqrt{n}}$.

This scheme has a parameter of its own, i.e. the initial step sizes. We chose to set $\sigma_i = 0.8$ since this value has been suggested as good in [1].

2) *De-randomised Self-Adaptation:* Ostermeier et al. proposed an alternative self-adaptive control scheme, labelled "de-randomised," specifically to alleviate problems with standard self-adaptation that occur with small population settings [15]. Applying de-randomised self-adaptation to $(\mu + 1)$ ON-LINE , updates are performed as follows:

$$\xi_i = \alpha; 1/\alpha \text{ with equal probability and } \alpha = 1.4$$
$$z_i = N(0, \sigma_i)$$
$$x'_i = x_i + \xi_i \cdot z_i$$
$$\sigma'_i = \sigma_i \cdot \xi_i^{\sqrt{1/n}} \cdot e^{\frac{|z_i| - \sqrt{2/\pi}}{n}}$$

Notice that de-randomised self-adaptation has no parameters.

The remaining three parameters of the $(\mu + 1)$ ON-LINE evolutionary algorithm are set to values that we have previously found to be good enough [7], i.e. $\mu = 6, \rho = 0.4$ and $\tau = 15s$. The only exception is the evaluation length for the gathering task where we chose to set $\tau = 120s$ due to the nature of the task and the form of the fitness function so as to allow controllers to reach more charger locations within the time limit.

V. Experimental results

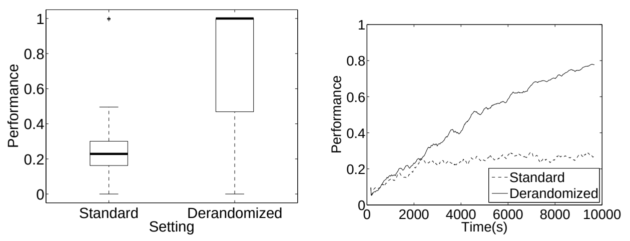
To compare the two σ control schemes on each of the three tasks we conducted six experiments. Each experiment consists of 100 independent runs with the same settings, using a different random seed. In each run we have a single robot running its own autonomous $(\mu + 1)$ ON-LINE evolutionary algorithm instance for a period of 10.000 (simulated) seconds.

To explain the plots showing the outcomes we must note that on-line evolution of controllers faces the complication that the *actual performance* of the robot matters. When a robot spends a lot of time evaluating poor controllers, it will perform very poorly on its actual task, even if the best individual archived in the population of its EA is of very good quality. Therefore, performance is measured over a sequence of activated controllers, not as the fitness of the best individual in the population at any moment.

The results of the experiments are presented in two ways. The mean performance during the final eight minutes of the evolution is shown in the usual boxplots. These show the end quality that has been achieved over the whole run. Additionally, we present curves exhibiting the development of performance over time. These provide information on the process of evolution.

The results for the fast forward task are shown in Figure 3. Performance of the de-randomised scheme is much higher when summarised over the last minutes of the evolution. Difference is statistically significant (t-test with 95% confidence). Evolution of performance over time shows that, though both strategies start evenly, standard self-adaptation stops improving at only 20% performance.

Results are similar for the phototaxis task (Figure 4). de-randomised self-adaptation performs better in average and has minimal spread (final performance is almost always



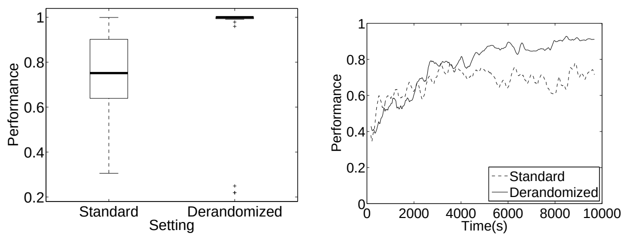
(a) Mean actual fitness during the final minutes of evolution averaged over all runs

(b) Development of actual performance vs. run time

Figure 3. Results for the fast forward task over 100 runs for the two σ control strategies.

close to 100%). Standard self-adaptation has a lower median and a large spread. Difference is again statistically significant according to a t-test with 95% confidence. As for the previous task, evolution of performance over time shows that, although the two schemes start comparatively, standard self-adaptation performance stops improving at an earlier point.

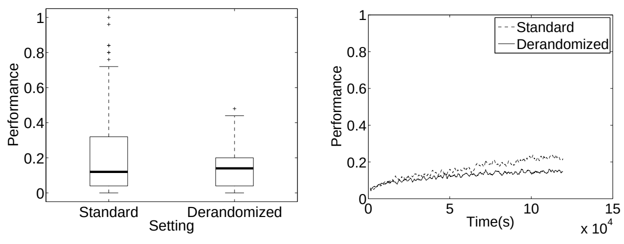
The results of the resource gathering task show a different situation. The end results for the two control strategies have very close medians around the performance level of 0.1, cf. Figure 5, left. This indicates that none of the strategies could consistently solve the given task. Data regarding the evolution over time (Figure 5, right) adds some nuance to this picture, showing a better average progress of the standard self-adaptation mechanism. However, this does not change the essence of our findings: both strategies fail to solve the task. The reason of this deviating algorithm behaviour lies most likely in the different problem characteristics. The first two problems have a quasi-continuous scale of evaluation and the actual value of the fitness function changes after each time instance (measurement). For the fitness values in the resource gathering task this is not true. The values are discrete and fitness values may remain constant over long periods, if the robot does not find any "food".



(a) Mean actual fitness during the final minutes of evolution averaged over all runs

(b) Development of actual performance vs. run time

Figure 4. Results for the phototaxis task over 100 runs for the two σ control strategies.



(a) Mean actual fitness during the final minutes of evolution averaged over all runs

(b) Development of actual performance vs. run time

Figure 5. Results for the resource gathering task over 100 runs for the two σ control strategies.

VI. Conclusions

In this paper we identified on-line, on-board evolutionary robotics as an area where self-adaptation is crucial. Based on common knowledge in evolutionary computing and our previous work in such robotic systems we argued that the parameters that require self-adaptation most are those associated with mutation. The question then is: what self-adaptation mechanism would be well-suited for the purposes of the kind of applications we have in mind, i.e., encapsulated evolution in autonomous robots (as envisioned in the SYMBRION project). To this end we implemented two self-adaptation schemes that have been previously offered in evolution strategies and performed an experimental comparison between them. This comparison was based on three different problems and we found that the de-randomised self-adaptation of mutation steps size clearly outperformed the standard self-adaptation mechanism on two problems. On the third problem, both mechanisms failed with equally bad median end results and slightly better average progress curves for the standard mechanism. Drawing general conclusions from these results is therefore not straightforward.

To this end, let us remark that generalising empirical results in evolutionary computing or, more generally, for heuristic algorithms is never straightforward [5], [9], but discussing methodological issues goes beyond the scope of this paper. For our present study we note that the choice of the three test problems is unbiased, that is, no attempt has been made to select tasks that would favour either of the self-adaptation schemes. In this sense, we have a fair sample. The sample size (three) is rather small, but comparing it to the rest of the literature it is not too small. All in all, we dare to articulate a slight preference for the de-randomised self-adaptation scheme, based on our results and the generic argument that it works better for small populations.

Future research is directed towards increasing our present sample size, that is, performing experiments on more problems. Furthermore, we will address the issue of selecting representative, or otherwise well-motivated test problems in order to facilitate generalisation of experimental results.

Acknowledgements

This work was made possible by the European Union FET Proactive Intiative: Pervasive Adaptation funding the SYMBRION project under grant agreement 216342. The authors would like to thank other partners in the SYMBRION consortium, Nicolas Bredèche, Selmar Smit, and Nicolaus Hansen for the inspirational discussions on the topics presented here.

References

[1] T Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK, 1996.

[2] Nicolas Bredeche, Evert Haasdijk, and A.E. Eiben. On-line, on-board evolution of robot controllers. In *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2009.

[3] A.E. Eiben, Evert Haasdijk, and Nicolas Bredeche. Embodied, on-line, on-board evolution for autonomous robotics. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, chapter 7, pages 361–382. Springer, 2010.

[4] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[5] A.E. Eiben and M. Jelasity. A Critical Note on Experimental Research Methodology in EC. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 582–587. IEEE Press, 2002.

[6] S. Elfwing, E. Uchibe, K. Doya, and H.I. Christensen. Biologically inspired embodied evolution of survival. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216, Edinburgh, UK, 2-5 September 2005. IEEE Press.

[7] Evert Haasdijk, A.E. Eiben, and Giorgos Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 3547–3553, Barcelona, 2010. IEEE Computational Intelligence Society, IEEE Press.

[8] Siavash Haroun Mahdavi and Peter J. Bentley. Innately adaptive robotics through embodied evolution. *Auton. Robots*, 20(2):149–163, 2006.

[9] J.N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[10] O. Kramer. Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 2010. On-line first, DOI: 10.1007/s12065-010-0035-y.

[11] F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, 2007.

[12] V. Nannen, S.K. Smit, and A.E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In G. Rudolph et al, editor, *Proceedings of the 10th international conference on Parallel Problem Solving from Nature (PPSN X)*, volume 5199 of *Lecture Notes in Computer Science*, pages 528–538. Springer, 2008.

[13] Ulrich Nehmzow. Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In C.G. Prince, Y. Demiris, Y. Marom, H. Kozima, and C. Balkenius, editors, *Proceedings of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, number 94 in Lund University Cognitive Studies, Edinburgh, UK, August 2002. LUCS.

[14] S Nolfi and D Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.

[15] Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. A derandomized approach to self adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1995.

[16] Anderson Luiz Fernandes Perez, Guilherme Bittencourt, and Mauro Roisenberg. Embodied evolution with a new genetic programming variation algorithm. *icas*, 0:118–123, 2008.

[17] H.-P Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[18] S.K. Smit and A.E. Eiben. Using entropy for parameter analysis of evolutionary algorithms. In T. Bartz-Beielstein and M. Chiarandini and L. Paquete and M. Preuss, editors, *Empirical Methods for the Analysis of Optimization Algorithms*, pages 211–224, Springer, 2009.

[19] Yukiya Usui and Takaya Arita. Situated and embodied evolution in collective evolutionary robotics. In *Proceedings of the 8th International Symposium on Artificial Life and Robotics*, pages 212–215, 2003.

[20] Joanne H. Walker, Simon M. Garrett, and Myra S. Wilson. The balance between initial training and lifelong adaptation in evolving robot controllers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(2):423–432, 2006.

[21] Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, April 2002.

[22] Steffen Wischmann, Kristin Stamm, and Florentin Wörgötter. Embodied evolution and learning: The neglected timing of maturation. In Francesco Almeida e Costa, editor, *Advances in Artificial Life: 9th European Conference on ArtificialLife*, volume 4648 of *Lecture Notes in Artificial Intelligence*, pages 284–293. Springer-Verlag, Lisbon, Portugal, September 10–14 2007.